

The Role of Design Pattern Decomposition in Reverse Engineering Tools

Claudia Raibulet & Francesca Arcelli

Università degli Studi di Milano-Bicocca,

DISCo – Dipartimento di Informatica, Sistemistica e Comunicazione

{raibulet, arcelli}@disco.unimib.it

Abstract: *The decomposition of design patterns into simpler elements may reduce significantly the creation of variants in forward engineering, while it increases the possibility of identifying applied patterns in reverse engineering. Key questions raise here: what should be design patterns decomposed in? How to recognize the application of design patterns by identifying their components?*

Currently, there are few forward and reverse engineering tools that exploit the decomposition of design patterns (e.g., FUJABA, SPQR). FUJABA is a forward and reverse engineering tool introducing sub-patterns to reduce the dimension of the design pattern catalog and the complexity of the elements searched in the source code, as well as to improve the detection algorithm. SPQR is an automatic tool for design pattern recognition, which introduces an elemental design patterns (EDPs) catalog and a rule set based on sigma-calculus through which EDPs are defined and composed into design patterns.

In this position paper we aim at focusing on the advantages and disadvantages of decomposing design patterns in sub-components and at introducing our research interests and projects related to this issue.

Introduction

The idea of decomposing design patterns into recurring elements has emerged both in the context of forward and reverse engineering. Such elements are called fragments [6], motifs [5], minipatterns [4], sub-patterns [10, 11], or elemental design patterns [13, 15]. As the variety of the names suggests there is little agreement what design patterns should be decomposed in. From the abstraction point of view, the sub-components of design patterns should be situated at the half-way between the source code and the high-level definition of design patterns. Sub-components of design patterns should reduce the generation of variants in forward engineering, and increase the rate of identifying applied patterns in reverse engineering.

Although decomposing design patterns into sub-components may improve significantly their detection process and results, there are few tools (as far as we know) that exploit this approach: FUJABA (From UML to Java And Back Again) [10] and SPQR (System for Pattern Query and Recognition) [14]. The reason of decomposing design patterns into sub-components in the context of the two tools is different, however both obtain significant results. FUJABA is a forward and reverse engineering tool exploiting *sub-patterns* [11] to reduce the dimension of the design pattern catalog and the complexity of the elements searched in the source code, as well as to improve the detection algorithm. It builds a

hierarchy of sub-patterns by assigning them a level number which is exploited by the detection algorithm to establish the order of applying transformation rules. SPQR is an *automatic* tool for design pattern detection. *Elemental Design Patterns* (EDPs) [13, 16], the sub-components of design patterns, play a central role in the context of SPQR. Extraction of information from source code (currently only for C++) is performed according to the elements EDPs are built of. The design pattern detection is reduced to the EDPs detection, while design patterns are expressed exclusively through EDPs. EDPs and their composition rules are expressed formally in terms of rho-calculus [17], which represents a subset of sigma-calculus [1] extended with new reliance operators. Rho-calculus inherits from sigma-calculus the type definition, object typing, and type subsumption concepts. Reliance operators are defined as direct, quantifiable expressions which indicate whether elements rely or depend on other elements and to what extent they do so. These operators indicate reliance on method invocation, field access, or generalization. Note that EDPs form a plain abstraction level, this meaning they can be detected independently of each other. A detailed comparison of these two tools together with the advantages they provide is described in [2].

Advantages and Disadvantages of Decomposing Design Patterns

Considering the FUJABA and SPQR approaches, we summarize the advantages sub-components provide to define and detect design patterns:

- sub-components are less complex than design patterns, hence also the rules their detection is based on are simpler;
- design patterns can be described *formally* (according to SPQR) as combinations of their sub-components;
- decomposing patterns into sub-components do not affect the flexibility of the first once; the variants problem impacts mainly on sub-components;
- sub-components represent an intermediate abstraction level between design patterns and source code; they can be considered also independently from design pattern in the context of reverse engineering due to their ability of incorporating design intents.

Disadvantages are related to the initial stage of this research issue, hence the identification and specification of the sub-components of design patterns and how these sub-components are combined to form design patterns. In our opinion, there are design patterns that can be formed by the same set of sub-components, which are combined in different ways leading to different semantics.

Current and Future Work

Considering the SPQR approach particularly interesting, we have extended it to software systems written in Java [3]. We have used the Recoder [12] framework to parse the source code and to generate its equivalent AST. Further, we have implemented a Visitor [7] to extract the information given as input to the OTTER theorem prover used by SPQR and to encode this information in a POML (Pattern Object Markup Language) file [17]. The output of our prototype called J2POML [3] is given as input to the theorem prover of SPQR, which provides a report with the detected design patterns.

Further, we are evaluating the idea of using the catalog of EDPs independently of the SPQR approach. This choice considers also that SPQR is based on a mathematical paradigm not commonly used by software engineers, and difficult to be understood and extended for other programming languages or new design patterns. Moreover, SPQR is not available for testing.

We have implemented a prototype called EDPDetector4Java [9] which is able to identify EDPs within the Java source code. Each EDP is detected through two main types of information: (1) the relationship between the method (*referrer*) which calls another method (*referred*), and (2) the relationship between the object which contains the referrer (*sender*) and the object which contains the referred (*receiver*). The analysis of these two relationships is described in detail by the authors of SPQR in [15]. The goal of SPQR has been to generalize the analysis for object-oriented languages. Our aim is to particularize the analysis for the Java language. Each method invocation which characterizes the behavior of the objects defining an EDP is determined by both the relation between the objects and the relation between the methods involved in the interaction. We have identified eight methods' and objects' properties we can exploit in the Java language to define and detect all EDPs. In our approach, EDPs are described through functions expressed in terms of canonical sums of product forms of the eight properties. Our approach is definitely simpler than that of the SPQR, but it pays in precision. The results show that six of the sixteen EDPs are uniquely identified through a single combination of the eight properties, five EDPs through two combinations, three EDPs through four combinations, one EDP through six combinations, and one through ten combinations.

Currently, we have concentrated our attention only on static analysis of the source code. The limitations of such an analysis are generated by the polymorphism of the object-oriented languages. Our approach uses an AST representation of the source code in association with information related to type expressions, reference resolutions, and cross-references. Considering only the static analysis, we cannot detect EDPs precisely, hence they have associated a degree of uncertainty as in the FUJABA approach. We aim at overcoming this aspect by considering also the dynamic analysis of source code to determine the run-time type of the *receiver* and the *referred*.

Further work will follow two directions: (1) introducing dynamic analysis of the source code in order to uniquely identify each EDP through one combination of the eight Java properties, and (2) the identification of design patterns through combinations of EDPs. For the first objective we are studying the CAFFEINE [8] approach. Further, we have implemented the EDP catalog into FUJABA [18] to perform a deep comparison between the sub-patterns of FUJABA and the EDPs, comparison which may lead to a unified design patterns' sub-components catalog

References

- [1] M. Abadi, and L. Cardelli, *A Theory of Objects*, Springer-Verlag, New York, Inc., 1996.
- [2] F. Arcelli, S. Masiero, C. Raibulet, and F. Tisato. A Comparison of Reverse Engineering Tools based on Design Pattern Decomposition. In *Proceedings of the Australian Software Engineering Conference*, Brisbane, Australia, March, 28th-31st, 2005, pp. 262-269

- [3] D. Bellinzona, "J2POML: Extraction of Information for Design Pattern Recognition from Java Source Code", University of Milano-Bicocca, Milan, Italy, November, 2004
- [4] M. Ó Cinnéide. *Automated Application of Design Patterns: A Refactoring Approach*. Ph.D Dissertation, University of Dublin, Trinity College, 2001
- [5] A. H. Eden. *Precise Specification of Design Patterns and Tool Support in Their Application*. Ph.D Dissertation. Department of Computer Science, Tel Aviv University, 2000
- [6] G. Florijn, M. Meijers, and P. van Winsen. Tool Support for Object Oriented Patterns. In *Proceedings of the 11th European Conference on Object-Oriented Programming*, Springer Verlag, Berlin, Germany, 1997
- [7] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: elements of reusable object-oriented software*, Addison Wesley, Reading MA, USA, 1994
- [8] G. Y. Gueheneuc, R. Douence, and N. Jussien. No Java without Caffeine: A Tool for Dynamic Analysis of Java Programs. In *Proceedings of the 17th IEEE International Conference on Automated Software Engineering*, Sempتمبر, 2002, pp. 117-126
- [9] S. Masiero. Design Pattern Detection in Reverse Engineering – The Role of Sub-Patterns. Master Thesis, University of Milano-Bicocca, Milan, Italy, October, 2004
- [10] U. Nickel, J. Niere, and A. Zündorf,. The FUJABA Environment. In *Proceedings of the 22nd International Conference on Software Engineering*, Limerick, Ireland, 2000, pp. 742-745.
- [11] J. Niere, W. Schäfer, J. P. Wadsack, L. Wendehals, and J. Welsh. Towards Pattern-Based Design Recovery. In *Proceedings of the 24th International Conference on Software Engineering*, Orlando, Florida, USA, 2002, pp. 338-348.
- [12] Recoder - <http://recoder.sourceforge.net/>
- [13] J. McC Smith. An Elemental Design Pattern Catalog. In *Technical Report TR02-040*, University of North Carolina at Chapel Hill, USA, December 10th, 2002.
- [14] J. McC. Smith, and D. Stotts. SPQR: Flexible Automated Design Pattern Extraction From Source Code. In *Proceedings of the 2003 IEEE International Conference on Automated Software Engineering*, Montreal QC, Canada, October, 2003, pp. 215-224
- [15] J. McC Smith, and D. Stotts. Elemental Design Patterns: A Link Between Architecture and Object Semantics. In *Technical Report TR02-011*, University of North Carolina at Chapel Hill, USA, March 25th, 2002.
- [16] J. McC. Smith, and D. Stotts. Elemental Design Patterns: A Formal Semantics for Composition of OO Software Architecture. In *Proceedings of the 27th Annual IEEE/NASA Software Engineering Laboratory Workshop*, Greenbelt, MD, 2002, pp. 183-190.
- [17] J. McC. Smith, and D. Stotts. Elemental Design Patterns and the Rho-Calculus: Foundations for Automated Design Pattern Detection in SPQR. In *Technical Report 03-032*, Computer Science Department, University of North Carolina at Chapel Hill, September 2003.
- [18] D. De Bortoli, and L. Conti. *Recognition of Elemental Design Patterns in FUJABA*. BSc Thesis, University of Milano-Bicocca, Milan, Italy, April, 2005.